.

Selective visual attention for *in vitro* neural stimulation

INI-508: Neuromorphic Intelligence

14.06.2021

Class Project Report

Simon Steffens

1. Introduction & Motivation

1.1 Biohybrid vision implant

The overarching project inspiring this work is the endeavour of building a brain machine interface that uses ectopic axons as electrodes. This approach is motivated by the limitations of current technology to deliver high-density stimulation in the brain with spatial resolution beyond large neural populations. While recording technologies have made considerable progress over the last decades, stimulation methods have not kept up. In medical applications, deep brain stimulation of basal ganglia has received a lot of attention over the last years, especially due to the remarkable improvements for patients suffering from Parkinson disease. Still, these systems suffer from a range of shortcomings that limit their utility in other settings: first and foremost, the spatial resolution of stimulation is limited to neural populations or entire nuclei, second, immunoreaction to the implanted electrodes causes complications in the long term, and lastly, these systems are limited in their adjustability post surgery, as voltage and pulse width are the only tunable parameters. Our biohybrid multielectrode array (bioMEA) aims to achieve stimulation at single-neuron resolution while simultaneously resolving the latent issue of biocompatibility encountered with implanted metal electrodes. Such single-cell resolution interfaces are most likely required for delivering high dimensional information, for example visual input. Our biohybrid interface will be implanted in the dorsal lateral geniculate nucleus (dLGN) restoring the visual input as depicted in Figure 1 below.

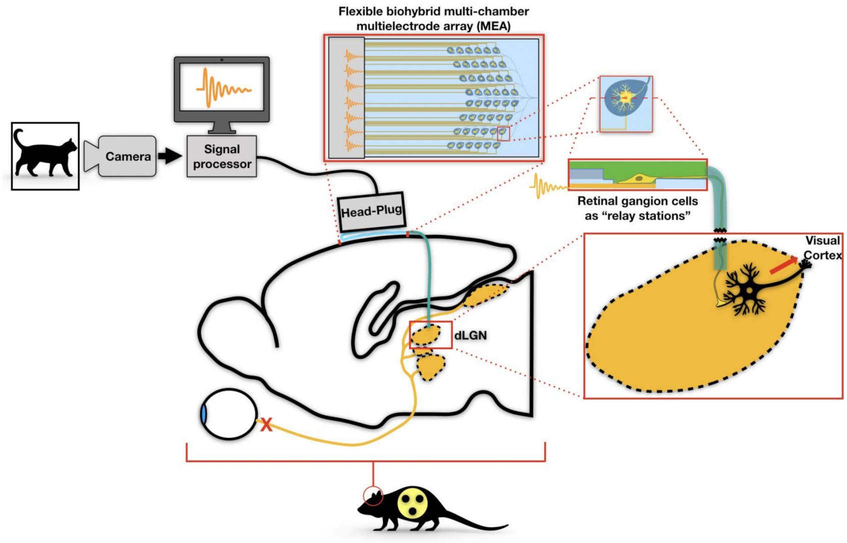

Flexible biohybrid multi-chamber multielectrode array (MEA)
Camera
Signal processor
Head-Plug
Retinal gangion cells as “relay stations”
Visual Cortex
dLGN
X

Figure 1: We prepare the implant *in vitro*, with axons already inside the structure. The device is then implanted under the skull such that ectopic neurons face down to receive nutrients from the brain surface; the guiding axon channel terminates in dLGN.

.

1.2 Signal processing requirements

Building this biohybrid interface involves a whole range of engineering challenges. One of the interesting problems to solve is the signal processing pipeline from a camera sensing the visual scene to delivering appropriate stimulation patterns to the relaying neurons growing on the implant. As the guiding channel is implanted in dLGN, one way to frame this problem is to think of replicating the natural input patterns the dLGN receives, using electronics. Retinal ganglion cells (RGC) innervating the dLGN transmit the output of the retinal circuitry, thus, the camera and signal processor may replicate the processing performed in the retina. This initial stage of computation encompasses a collection of impressive features. The central functions of the retina include both local and global gain control and initial feature detection: RGC receptive fields are tuned for bright spots on dark background or dark spots on bright background (spatial dimension), and additionally respond strongly to changes in the input (temporal dimension). On top of that, a replication of the retinal output would require sensors with four different wavelength selectivities, superimposed center surround receptive fields of different sizes, a distribution of light sensors resembling the fovea and macula, and finally an output bandwidth of 1.2 million (≈1024x1024). Attempts to build cameras that approach this impressive specsheet have been made since the advent of neuromorphic engineering (Mahowald & Mead, 1991). One of the more recent silicon retinas is the dynamic vision sensor (DVS/ DAVIS) (Brandli, 2014) pioneered by Tobi Delbruck. While the DVS lacks many of the retinal features, it is very capable of capturing temporal differentials in illumination. Log-intensity changes above a certain threshold can trigger ON-, (pixel input became brighter) and OFF events (input became darker). This yields an asynchronous event sequence for each pixel where an ON event may be interpreted as an ON-center RGC without OFF-surround and with strict specificity for differentials in the input, i.e. in contrast to RGCs, a constant bright spot will not trigger a sequence of events/ spikes. This discrepancy may however not be detrimental as saccades naturally move the visual scene at a high rate which creates differential brightness.

It seems intuitive to look into neuromorphic cameras like the DVS and signal processing chips like Dynap-SE for this vision implant because, here, the primary representation of information is fundamentally shared with biological neural systems: asynchronous spikes/ events. The pipeline from visual sensing to *in vitro* stimulation does not require advanced processing to convert synchronous frames into asynchronous waveforms. Another argument for relying on neuromorphic hardware is that the timescales at which these systems are made to operate match the timescales of natural stimuli and those of biological neurons. It might be difficult to produce these dynamics with an algorithm. Another advantage of neuromorphic hardware is the noise emerging from physics of semiconductors. When interfacing with real neurons, the noise arising from emulating neural dynamics may indeed be a feature, as it creates a match to the inherently stochastic firing of biological neurons. In conclusion, the DVS and neuromorphic chips present a promising direction for the visual signal processing required for this implant in the long term.

.

1.3 Motivation behind implementation

This project explores potential neuromorphic visual signal processing for our biohybrid vision implant. It should be noted that this is not an attempt to engineer the final system. It is rather aimed at familiarizing oneself with the problem to solve. The results from this project may also prove useful for ensuing grant applications, showcasing know-how with neuromorphic hardware and initial results for delivering real visual input to the neurons on the implant.

Since this project was quite time constraint, one had to settle on reasonable goals, which is why this work is limited to prerecorded DVS data that was fed to the Dynap-SE chip. Ideally, we would have designed a PCB that connects a DVS to an FPGA to the Dynap-SE2 and finally to a dish of cultured neurons placed in a portable incubator. Still, by feeding in pre-recorded event data we were able to explore the elementary step of processing the DVS output on the Dynap-SE.

First, we needed to define the overall goal of the signal processing pipeline. As outlined above, the initial idea of this project was to reproduce the biological dLGN input as close as possible. For this, the plan was to complement the 240x180 DVS output with spatial center-surround filters in post processing, creating stimulation patterns closer to those observed in real RGCs. The intuitive attempt to add spatial filters resembling center surround receptive-fields was made by one of Tobi Delbruck's postdocs, Dennis Gohlsdorf around 2013, but this was to no avail. Considering the bandwidth limitations of the integrated FPGA and the limited number of neurons on the Dynap-SE chip, we moved away from this idea.

Instead of substituting retinal processing, one could also imagine deviating from the output of the biological retina. This especially makes sense in the light of limited bandwidth in our biohybrid implant. Currently, the device is limited to 64 independent channels, resulting in a potential resolution of 8x8. One could frame the problem as *How do we provide maximum utility for carriers of this implant when we are limited to 64 channels?* From this perspective it seems reasonable to perform processing that goes beyond the formation of ON/OFF center-surround receptive fields and deliver richer information. Indeed, this ended up being the pursued approach.

.

1.4 Project outline

The events of the DVS are downsampled to a resolution of 8x8. This 8x8 input drives an 8x8 WTA network that selects for a dot-like stimulus blinking at 60 Hz. In WTA fashion, other stimuli, for example edges created by a moving visual scene are filtered out, resulting in selective visual attention for blinking stimuli. This selection for fast blinking stimuli may not seem in line with *maximizing utility* directly, however it could prove useful in a specific experimental setting. First *in vivo* experiments confirming functional connections between the ectopic neurons and dLGN would simply involve arbitrary stimulation patterns associated with some reward/ learning task. A far more impressive demonstration would be a learning task linked to a blinking LED, where the implant is actually connected to a head-mounted DVS. In this scenario, the WTA attention mechanism could reduce the complexity of a moving visual scene input arriving in dLGN to facilitate perception of the stimulus. The WTA network connects to a population of stimulating silicon neurons that produce voltage pulses suitable for *in vitro* neural stimulation. Of course, this system could also be tested in an *in vitro* setting, where cultured neurons innervate a tissue piece. Due to time constraints the generated stimulation patterns could unfortunately not be tested in practice.

.

2. Codebase overview

All the code of this project is available for open source use on github LoaloaF/dvs2dyn2neurons. Care was taken to make the code readable and extendable. The codebase should be useful for anyone getting started with simple DVS data processing and Dynap-SE programming using the new `samna` API. Much of the implemented functionality is already available in specific packages, especially from Davide Scaramuzza's lab. However, for getting started, their tools can be overwhelming and overly rich in the features they offer. That is were we see the value of this codebase.

Two different methods are provided for loading in DVS data. First, `vid2events.py` can be run to simulate event data from a frame-based video. Second, and the default way of loading event based video data into `numpy` is to read the `.aedat4` files DVS cameras produce. This functionality is offered in the script `process_aedat.py`. The required libraries for these two input loading scripts are `esim_py`, `dv`, and `opencv`. To run this code, an environment file named `dvs.yml` is included as well.

The `numpy` array returned by the input-loading functions is further processed in the `create_stimulus.py` script. This is also the location where all the data parameters are set, for example the general name of the dataset, the time interval to cut etc.. As the file name indicates, the raw event sequence is converted to a stimulus pattern suitable for the FPGA on Dynap-SE. This includes visualization on the effect of preprocessing on the data, for example event rates and inter spike intervals. The visualizations include videos that are encoded with `ffmpeg` or `imagemagick` wrapped around `matplotlib's` animations API. Video encoding frequently caused errors, which is why `create_stimulus.py` also has its own environment, `vid_render.yml`.

The output of `create_stimulus.py` is used by the `samna_network.py` script, which is run on the zemo server where the Dynap-SE blue box is connected. The stimulus is loaded from `.npy` array, possibly sampled down to fit the FPGA buffer size of 32500, and loaded onto the FPGA. Subsequently, the network is build, run for the length of the stimulus, and the spike monitor output is saved. The code has implements the option for iterative parameter testing.

For convenience, the repository also contains a bash script that `rsync`s the local code and data directories, with the remote ones on zemo. Executing `./run_on_zemo.sh` will first `rsync`, then run the `samna_network.py` on zemo. Once the python code has terminated, the remote data directory is `rsync`ed with the local one. Lastly, the `viz_dyn_output.py` script is run to visualize the Dynap-SE output. One drawback that we we were not able to solve was to flush the `print()` outputs dynamically during remote code execution. This output would only appear once the script terminates, a potential inconvenience for extensive parameter search. To solve this, there is also the `to_zemo.sh` script, which only `rsync`s the remote directories, followed by `sshpass`ing to zemo. There, one can manually execute the `samna_network.py` script to obtain the dynamic output of the code.

.
.

3. Results

3.1 Preprocessing

3.1.1 Simulated event video

As described above, the final result of this project relied on prerecorded DVS data. At first however, we used the python-based event simulator `vid2e` (Gehrig, 2020) for simplicity and fast testing of different stimuli. Although the publishers original motivation was generating event-based training data for deep learning, the package is also highly useful for anyone who wants to create event videos without a DVS camera. The `EventSimulator()` function takes in a video file, a frame timestamp file, and a collection of 5 parameters to generate a `numpy` array with timestamps, pixel coordinate and polarity. The implementation for this project generates the timestamp file automatically from frame counts and FPS in the recorded video. On top of that, the option for cutting the video is provided. All this is code is located in `vid2events.py`. Below is an example of the simulated events (or see `./figures/simulated_events.mp4`). These were generated from a webcam-recorded stimulus where an LED blinks at 20 Hz and the camera is moved slowly, mimicking head or eye movement.

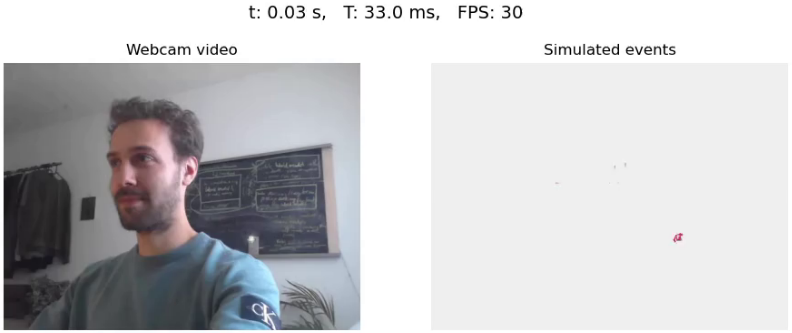


Figure 2: Simulated events using the esym_py package. The video was recorded on conventional 640x480 webcam at 30 FPS. The parameters used to simulate the event-based video are listed below.

```
    # parameters defining output of EventSimulator()
    constrast_thr_p = 0.4
    constrast_thr_n = 0.4
    refractory_period = 0.0001
    log_eps = 1e-3
    use_log = True

```

Some effort was invested to find the best simulation parameters, but the obtained results remained modest. Specifically, the LED blinking was not properly converted into the expected output of ON-, and OFF events alternating over the LED region. Instead, the slow camera movement dominates the output in that a small trailing ON-bar appears where the LED is located. This is the general pattern we expect for non-blinking bright objects that move in front of a dark background. To discriminate the blinking LED from the residual visual scene however, we prefer a qualitatively different visual input for the region where the LED is located. It may very well be the case that the webcam's FPS is the bottleneck here.

.

3.1.2 DVS stimulus recording

With the results from above, we decided to record event-based video data with a real DVS. The DAVIS sensor (180x240) linked to the DV software was used to record three stimuli of increasing discrimination difficulty. For this recording, we increased the LED blinking frequency from 20 Hz to 60 Hz to reduce the difficulty of the discrimination. To generate the stimulus, a simple `matplotlib.animation` script (below) with controllable frequency and circle size was used. We placed a laptop running the stimulus in the visual scene and started recording with the DAVIS.

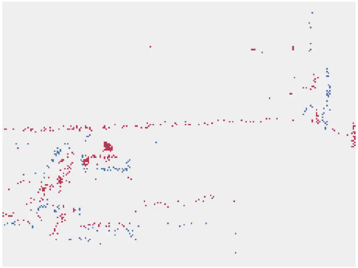
```python
from matplotlib import pyplot as plt
from matplotlib import animation

def LED(hz, size):
  # displays an animation of a circle alternating in color between black
  # and white on grey background.
  fig, ax = plt.subplots(figsize=(19,11), facecolor='gray')
  fig.subplots_adjust(left=0,right=1,bottom=0,top=1)
  ax.axis('off')
```

```python
    ax.set_facecolor('gray')

    on = ax.plot(1, .5, 'o', markersize=200*size, color='white')
    off = ax.plot(1, .5, 'o', markersize=200*size, color='black')
    ani = animation.ArtistAnimation(fig, [on, off], interval=1000//hz, blit=True
     )
    plt.show()

  def main():
    hz, size = 60, 1
    LED(hz=hz, size=size)

  if __name__ == '__main__':
    main()

```

As outlined in the motivation section above, we aim to create stimuli that are realistic for a potential carrier of our biohybrid vision implant. One relevant property of natural visual input, is that the visual scene is constantly moving either through head movement or eye movement (saccades). To superficially reproduce this, it seems reasonable to create a stimulus where the camera itself is moving. Two of the three stimuli, `blink_slow` and `blink_fast` follow this approach, where, as the name suggests, `blink_slow` was recorded with slow camera movement, `blink_fast` with rapid camera movement. The third stimulus was recorded with a static camera but with a moving person in the background to check if the network generalizes to an altered setting. Below are the three recorded datasets (or alternatively see `./figures/*recording.mp4`).

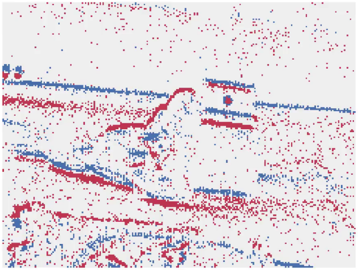
DVS_slow_motion

t: 0.10 s, T: 100.0 ms, FPS: 10, polarities: [both on & off events]



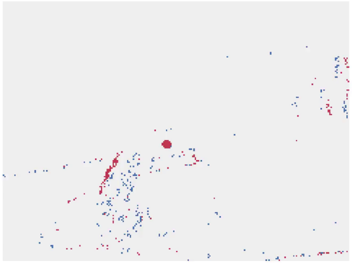
DVS_fast_motion

t: 0.10 s, T: 100.0 ms, FPS: 10, polarities: [both on & off events]



DVS_bg_motion

t: 0.10 s, T: 100.0 ms, FPS: 10, polarities: [both on & off events]



.

3.1.3 Downsampling to 8x8

The first major step on the offline data preprocesing pipeline is to downsample th DVS data from a resolution of 180x240 to 8x8. This output resolution is dictated by the limited number of channels, 64, in the biohybrid vision implant. Although not an issue in this application, the number of neurons on the Dynap-SE chip may limit the image resolution we can process.

Downsampling is performed in a reasonably simply manner.